REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	19 December 1996	Final Report

4. TITLE AND SUBTITLE

Wavefront sensing for liquid crystal adaptive optics

5. FUNDING NUMBERS

F6170897W0018

6. AUTHOR(S)

Dr. James Gourlay

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The University of Durham
Old Shire Hall
Durham DH1 3HP
United Kingdom

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SPC 97-4006

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

This report results from a contract tasking The University of Durham as follows: The contractor will develop and test software to perform the following tasks: a) interface with a Dalsa CCD Camera in order to read data from a Shack-Hartmann lenslet array. b) Calculate the centroid positions of an 8x8 array of sp rom a Shack-Hartmann sensor. c) Calculate the local wavefront gradients from the centroid positions. d) Use the data from c) to reconstruct the Zernike coefficients for the wavefront. e) Calculate the phase shifts to be applied t exagonal array of 127 pixels. f) Convert the phase shifts into the voltages that are necessary to actuate the liquid crystal - spatial light modulator.

14. SUBJECT TERMS

Physics

15. NUMBER OF PAGES

43

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Liquid Crystal based Adaptive Optics System: Software v1.0

Final Report for project:

Wavefront Sensing for Liquid Crystal Adaptive Optics

Dr James Gourlay

December 19, 1996

Contents

1 Introduction 3

2 Design Philosophy 3

2.1 The problem . . . 3
2.2 The wavefont sensor . . . 3
2.3 The full system . . . 3
2.4 Computational Hardware . . . 3
2.5 The optical setup . . . 3

3 Description of Hardware 4

3.1 Processor board . . . 4
3.2 Wavefront sensor camera . . . 4
3.3 Lenslet Array . . . 4

4 Software Description 5

4.1 Requirements . . . 5
4.2 Functionality . . . 5

5 Calculations Performed 7

5.1 Boxing . . . 7
5.2 Centroiding . . . 7
5.3 Calculation of Zernike Mode Coefficients . . . 7
5.4 Control loop -Integration . . . 7

6 Software Development 7

6.1 Compilation of c40 code for DSP . . . 8
6.2 Compilation of code for GSP . . . 8
6.3 Software testing . . . 8

7 Operation Proceedures 8

7.1 Alignment . . . 9
7.2 Setting the system gains . . . 9
7.3 Closed-loop operation . . . 9

8 Conclusions and Future Recommendations 9

9 References 9

A Hardware List 10

B Software List 11

C Zernike Modes 12

D Sotware Listings: test_gsp.c 13

E Software Listings: test_dsp.c 24

1 Introduction

The aim of this report is to describe the design, development and subsequent testing of software to operate a liquid crystal (LC) based adaptive optics system. The work was performed by James Gourlay, Dept. of Physics, University of Durham, inconjunction with Phillps Labs.

2 Design Philosophy

2.1 The problem

The purpose of the software development is to produce a closed-loop adaptive optics system comprising of a Shack-Hartmann wavefront sensor and a LC phase control device. The system will have the ability to track real-time turbulence, commensurate with that expected from the atmosphere, so that we can improve atmospheric seeing. Static aberration can also be corrected for. The system will provide a low-cost laboratory based development tool, plus allow preliminary "real-life" experiments e.g. at telescopes.

2.2 The wavefont sensor

The Shack-Hartmann wavefront sensor consists of a number of lenslets which sample the optical wavefront. The tip-tilt of the lenslet spots give an indication to the local wavefront slope, and from this information, the wavefront shape can be estimated by interpolation. The first main design consideration is the representation for the wavefront. Zernike functions are used to describe atmospheric aberration, and provide a convenient means of representing our data in the system. The Zernike functions or modes, are independent, and should therefore simplify any filtering and testing, as each can be regarded individually.

2.3 The full system

Once the zernike mode coefficients are calculated for the wavefront, they can be used to modify the correction applied to the liquid crystal device. To make the system more robust to device variations and errors, we plan to run in a closed-loop operation. Therefore, the measured Zernike mode coefficient must be filtered before being applied to the liquid crystal correction device. By experimentally adjusting the closed-loop gain for each mode, we can optimise system performance and improve overall aberration tracking and closed-loop bandwidth.

2.4 Computational Hardware

This will be a PC host computer based system, with an F/64-DSP processor board, accessing CCD camera information, performing image processing operations and calculations to produce the measured Zernike mode coefficient values. The system will also run the control algorithms required to operate in closed-loop. Software to generate Zernike modes on the LC device has already been produced by Phillips labs, and this will be incorporated into the control loop.

2.5 The optical setup

Setting up such an optical system is very complicated, and the software will provide various diagnostic tools to aid this process. For example, on screen display of measured modes allows the

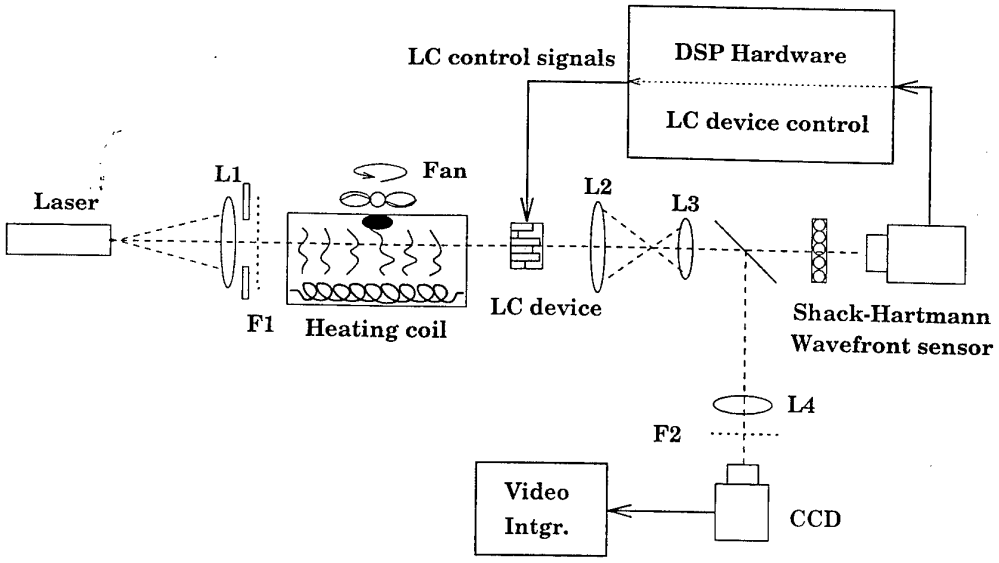

Figure 1: The liquid crystal based adaptive optics system

static errors in the optical system to be minimised. Figure 1 gives an indication of the optical system setup used in experiments.

3 Description of Hardware

3.1 Processor board

The Coreco F/64-DSP board was selected for processing the data. It has an 8-bit frame buffer, a high performance TMS320c40 DSP chip and can digitise the DALSA camera data which is output at 16 MHz. The board comes with full software support including Oculus image processing functions. A windows based demonstration program F64Pro is also included, which allows for simple testing of the hardware.

3.2 Wavefront sensor camera

The camera used in the wavefront sensor was a DALSA CA-D1-0128, adjusted to operate at a data rate of 16MHz. With 128 by 128 pixels, the camera can operate at a frame rate of 838Hz. The camera is not really suitable for low-light applications, and should be replaced with a more sensitive (and hence more expense) camera for low-light experiments. The sensor dimensions are 2.90mm across the diagonal. Figure 2 shows how the camera is configured with the DPS board and associated hardware.

3.3 Lenslet Array

The lenslet array used has lenslets centred on a hexagonal grid, matching the LC SLM. It was produced by AOA, and supplied by Phillips Labs. 19 lenslets were used to sample the wavefront.

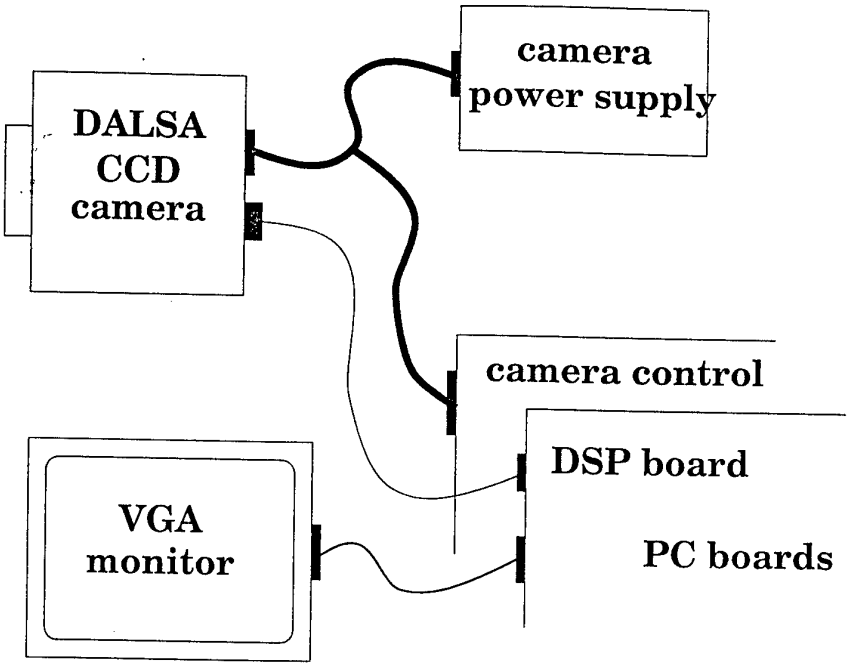

Figure 2: The configuration of the DSP system hardware

4 Software Description

The software has been designed in two separate parts. Software for the DSP processor written in Texas Instruments c40 code, and software in Borland C++ for the host PC. In general, for fast operation, the DSP handles the data processing concerned with accessing the camera frame buffer store, and processing of the camera images. A reduced data set, in our case the Zernike mode coefficient values, are accessed by the host computer using the techniques described in the Coreco example code.

4.1 Requirements

To allow development and use of this software, some supplimentary code is required. All the software supplied by Coreco for the F/64-DSP board should be installed, including the Oculus F/64 device Driver, Oculus F/64-DSP Toolkit, F64Pro executable, ODX Programmers Toolkit, and the Oculus Driver Command Interpreter, as per instructions supplied by Coreco Inc. Also, a Texas Instruments C40 parallel compiler is required and a standard C compiler such as Borland C++.

4.2 Functionality

An indication of the the basic software functionality is shown in the figure 3. The software has two basis features, and initialisation, and then a control loop. User interface is achieved through a menu of options, which can be selected and options changed as the code is running. Some of the functions performed will be described in the next section.

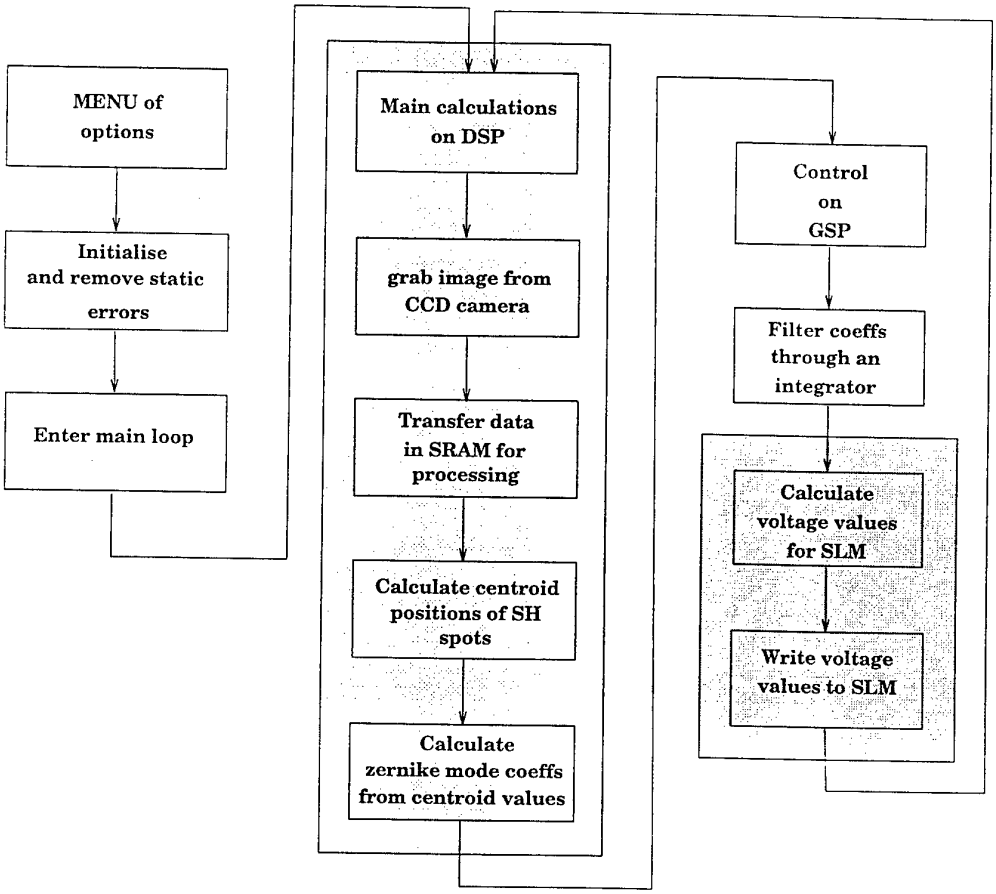

Figure 3: The software functionality

5 Calculations Performed

5.1 Boxing

As the lenslet spots are expected to be in certain positions in the frame buffer memory, a user defined sample box size is used to determine what region around each spot zeroth location is scanned by the centroiding algorithm.

5.2 Centroiding

The centroiding algoirthm is possibly the simplest imaginable. In one dimension, for 1 to m pixels, where the intensity of the n^{th} pixel is I_n, the deviation of the spot centre is the sum of all of the spot positions multiplied by their intensities, all divided by the total intensity in the sample region i.e.:

$$position = \frac{\sum(I_n \times n)}{\sum I_n}$$

For noisy data and less well defined spots, this technique will have to be improved. After site testing at Sunspot solar observatory, a simple bias removal was added to this algorithm to increase the accuracy with low contrast data.

5.3 Calculation of Zernike Mode Coefficients

The calculation of the Zernike mode coefficients is performed by a standard matrix multiply with an interaction matrix which has been calculated off-line in MATLAB and the result (which is constant in our experiment) is included in the DSP code. The interaction matrix effectively transforms the wavefront data, represented by local tip-tilts (what is measured) and represesnts it in terms of Zernike mode coefficients i.e. 2 × m tip-tilt measurements (one for x, and one for y) is transfomred into n Zernike modes by multiplication with a 2 × m by n matrix.

5.4 Control loop -Integration

If one assumes that the LC correction device is the slowest component in the loop, and that it has the form of a low-pass filter, standard closed-loop control theory analysis suggests that to optimise and stabilise our system we must introduce an integration function into the loop. This has the form:

$$y_n = K(y_{n-1} + x_n + A * x_{n-1}),$$

where K is the gain, y_n is current output, y_{n-1} is previous output, x_n is current input, x_{n-1} is previous input, and A is a coefficient determined by the closed-loop bandwidth.

This has to be modified after experimentation when the speed of the system can be verified and was investigated in conjunction with Phillips Labs.

6 Software Development

The software for the system was developed along the lines of the example code supplied by Coreco to run there processor board.

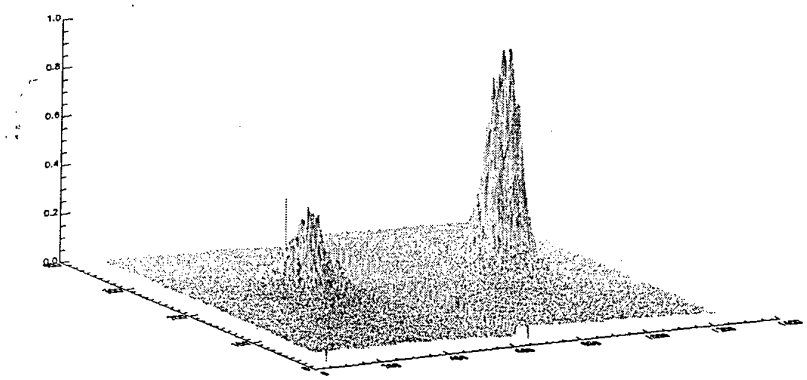

Figure 4: An example of closed-loop correction (before and after) on simulated atmospheric turbulence, the Strehl ratio is increased by 2.5 times.

6.1 Compilation of c40 code for DSP

This requires the Texas Instrument c40 parallel compiler. As we are using only a single c40 processor, it can be used much as any other C compiler. The code is based on the example functions called user.c in the Correco documentation.

6.2 Compilation of code for GSP

This can be written in any standard C. We have written the code in Borland C++, as the SLM control software also runs under this version. The code is based on the example functions called userodx.c in the Coreco documentation. The software runs in a loop, with a menu to allow user interaction.

6.3 Software testing

The software was fully tested at Durham University, and has undergone successful site testing both at Phillips Labs and Sunspot Solar Observatory. The full system frame rate was optimised to run at 62 Hz (i.e. frame grabbing, filtering, centroiding, control algorithm - integrating and writting voltages to LC SLM), giving a closed-loop bandwidth of between 5 - 10 Hz. Software optimisation was performed in conjunction with Phillips Labs. Running in closed-loop, the full system corrected for simulated atmospheric turbulence in laboratory experiments, producing an increase to the Strehl ratio of 2.5 times. The output Point Spread Functions from the system were integrated over approximately 10 seconds on a truth/science camera, and an example of the results for before and after the correction was applied is shown in figure 4. These results are to be published.

7 Operation Proceedures

This section describes the operation proceedure of the system using the software described above. We assume that the optical system is setup in accordance to figure 1, and the hardware is configured as in figure 2.

7.1 Alignment

After reseting the PC, one initialises the system and loads the DSP code by running *lo.bat* from a DOS prompt. One then runs *test_gsp.exe*. From the menu, select option 1, system alignment. This runs only a wavefront measurement and display of measured Zernike coefficients, but no filtering or correction (i.e. the LC SLM has no phase pattern). One can now adjust the lenslet array and camera, so that the lenslet spots orientate with the white alignment crosses displayed on the VGA monitor. This can firstly be performed visually, and then using the Zernike mode measurments, particularly tip, tilt and defocus, to tune this finely. Once aligned, there is still likely to be some residual mode measurement, these static values can be removed from dynamic measurements by selecting option 2 in the menu and "fudging".

7.2 Setting the system gains

For each Zernike mode corrected in the system, there is an individual control algorithm with a gain value. The gains must be experimentally obtained. The proceedure involves only correcting for one single mode (all others have gain set to zero), and slowly increasing the gain value (a define statement at the start of test_gsp.c), until the system just about starts to oscillate, when a step aberration is introduced. One can then repeat the procedure for the other modes.

7.3 Closed-loop operation

One can then select option 5, which will run closed-loop correction with no diagnostics written to the PC or VGA screens. Other options include correcting with diagnostics, and outputing measured mode values to to the file *out.dat*.

8 Conclusions and Future Recommendations

The software gives a solid frame work for future system developements and will allow simple modification for more powerful processing boards. It is recommended that more powerful processing power be included in future hardware considerations. Coreco supply a newer version of board and can supply more processor modules.

9 References

1. F/64Pro User's Guide, Edition 1.0, revision 1, Coreco Inc.
2. Using the Oculus F/64 DSP Toolkit, Coreco Inc.
3. The Oculus F/64 Frame Grabber, User's Manual Edition 1.0, Revision 2, Coreco Inc.
4. CA-D1 Camera Users Manual, Dalsa Inc.

A Hardware List

- DALSA CA-D1-0128 CAMERA (16MHz)
- F/64-DSP, Image Processor with 2MB VRAM, 1MB DRAM.
- E1A-422-12 BIT DIGITAL INTERFACE

B Software List

List of software delivered to Phillips Labs.

Basic code:

- **test_gsp.c** the code to run on the PC, compile with Borland C++ and link with odxcall.c and zern_slm.c.
- **test_gsp.exe** the executable
- **test_dsp.c** the basic code for the c40 DSP processor, compile with cc.bat, and load onto processor board with lo.bat
- **test_dsp.obj** the objective for linking.

Advanced level:

- **fast_gsp.c** a faster version of test_gsp.c, developed inconjunction with Phillips Labs.
- **fast_dsp.c** a faster version of test_dsp.c, developed inconjunction with Phillips Labs.

Compilation files:

- **cc.bat** batch file for compiling the c40 code
- **cr.bat** a batch file for compilation with Microsoft C.
- **lo.bat** links the c40 code and loads it onto the processor board.
- **f64dsp.cmd** this file tells the c40 linker which .obj file to link to (test_dsp.obj) and then loads it to the DSP board.

Extra:

- **zern_slm.c** must be linked in with GSP code (test_gsp.c) with Borland C++ compiler.
- **odxcall.c** must be linked in with GSP code (test_gsp.c) with Borland C++ compiler.
- **d128_16.vid** file containing camera parameters supplied by DALSA. use config or wconfig so that board looks for correct custom camera.
- **hex.dat** Co-ordinates of the lenslet positions, read into test_gsp.c

C Zernike Modes

Zernike mode functions in terms of r and θ (non-normalisied):

- mode 0: z = 1 (piston)
- mode 1: z = r cos θ (tip)
- mode 2: z = r sin θ (tilt)
- mode 3 : z = 2 r^2 - 1 (defocus)
- mode 4: z = r^2 sin 2 θ (astigmatism)
- mode 5: z = r^2 cos 2 θ (astigmatism)
- mode 6: z = ($3r^4$ - 2r) sin θ (coma)
- mode 7: z = ($3r^4$ - 2r) cos θ (coma)
- mode 8: z = $6r^4$ - $6r^2$ + 1 (spherical)
- mode 9: z = r^3 sin 3θ
- mode 10: z = r^3 cos 3θ

D Sotware Listings: test_gsp.c

```
/*************************************************************************/
/*                                                                       */
/*      LIQUID CRYSTAL BASED ADAPTIVE OPTICS SYSTEM                      */
/*      Shack-Hartmann  and hex127 SLM control software                  */
/*      use in conjunction with test_dsp etc.                            */
/*      J.Gourlay  1 Nov 96                                              */
/*************************************************************************/


/********include files for microsoft C version**********************/

#include "zern_127.h"              /* include for hex127 slm software */

#include "c:\odx\cc\odx.h"         /* Definition of each ODX function */
#include "c:\odx\cc\opr.h"         /* Symbol definition for each OPR */
#include "getarg.h"         /*Changed Location!!!*/

/* User Defined ODX Function Call   */
/*
 * wlib: indicates library to use ( must range from 0 to 15 inclusively)
 * we use 0
 * wop:  indicated which function to execute ( must range from 0 to 2047 inclusively)
 * we use 0
 * buf:  pointer to user function structure
 */

/**********program defines*************************************************/

#define USER_DSP_FUNCTION( wlib, wop, buf) ((*odx_send)(47,
(unsigned short) ((wlib << 11) | (wop & 0x7ff)), (void far *)
        (buf)), (*odx_rcvi)())

#define POINTS 19      /* number of lenslets */
#define SCALE 85
/* scale/magnification of lenslet centres fitting into 128 by 128 grid */
#define X_CENTRE 64    /* to position CCD array with lenslets */
#define Y_CENTRE 65
#define MODES 12       /* number of modes corrected */
#define GAIN1 0.25     /* gains for each mode       */
#define GAIN2 0.25
#define GAIN3 0.25
#define GAIN4 0.25
#define GAIN5 0.25
#define GAIN6 0.25
#define GAIN7 0.25
#define GAIN8 0.25
#define GAIN9 0.25
#define GAIN10 0.25
```

```
#define GAIN11 0.25
#define GAIN12 0.25

#define MAX_NUMBER_OF_TERMS 36
/* max zernike terms/modes for slm software */

#define N_LOOPS 30              /* for slm software */



/* Structure for User defined Function #0 */
/* Don't forget: For the TMS32C040 all C data types are 32 bits */
/* All parameters of your user defined structure should use the long or
   unsigned long data types.  If not, you will have to unpack your argument
   inside your user defined function implement on the TMS320C40.
   Also, since the TMS320C40 uses a different representation to implement
   the floating point data type you should take great care when passing them
   to the DSP.
 */



/* structure for data transfer to coreco board */

typedef struct
   {
   unsigned long spotson;
   unsigned long xpos;
   unsigned long ypos;
   unsigned long dir;
   unsigned long grl;
   unsigned long pfbpage;
   unsigned long far *gArray;
   unsigned long far *gArray2;
   unsigned long far *gCoeffs;
/* data transfer using floats */
   float far *gCoeff2;
   float far *gCentres;

   }  USERFCT0;


/* global variables - bad programming but what the hell.... */

   int initialise = 0;    /* initialise */
   int initial2 = 0;      /* fudge */
   int diagnostics = 1; /*diagnostics on*/
   int correct = 1;       /* correction on */
   int file_out = 0;      /* file_out on */
```

```
/* function prototypes */

void OPTIONS(void);


void main()
   {
/* program variables */

   int counter;  /* a counter to count frames for timing processing */
   char szStr[ 80], c ='n';
   int errno;
   int option= 0;
   int bgrab= 1;
   int breset= 1;
   int loop;
   FILE *fp;
   FILE *fpout;
   int i,number,delay,delay2;

   /* old transfer variables (not really used) */
   static unsigned long Array[POINTS*2]; /* coords of lenslet centres */
   static unsigned long Array2[POINTS*2];
   static unsigned long Coeffs[MODES];  /* old array with zernike mode values*/

   /********************************************/

   float fudge[MODES] = {0,0,0,0,0,0,0,0,0,0,0,0} ;
/* fudged coords of lenslets */

   static float Coeff2[MODES];
/* array with measured zernike coeffs from SH */
   static float Centres[POINTS*2]; /* lenslet centres */
   float hex_array[POINTS*2]; /* lenslet centres */


   /*******variables from zern_gen.c*************/

   float coeffs[MAX_NUMBER_OF_TERMS] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
/* array with zernike coeffs for SLM */


/* Hex array coords are read from file hex.dat */

   printf("Inputing hex array coords\n");
   if((fp = fopen("hex.dat","rt")) != NULL)
      {
```

```
/* n.b. hex.dat in polar coords, so convert to cartesian  */
      for(i=0; i<POINTS*2; i=i+2)
         {
          fscanf(fp,"%d %f %f \n", &number, &hex_array[i], &hex_array[i+1]);
Array[i] = (unsigned long) (hex_array[i]*cos(hex_array[i+1])*SCALE + X_CENTRE);
Array[i+1] = (unsigned long) (hex_array[i]*sin(hex_array[i+1])*SCALE
                                  + Y_CENTRE);
        printf("%d %d %ld %ld\n", i, number, Array[i], Array[i+1]);
/* convert and put into Array for transfer to DSP */
         }
      fclose(fp);
      }
/* open diagnostic file out.dat */
   fpout = fopen("out.dat","wt");

 /**************Initialise Hex 127 SLM*************************/
   lc_init();

/* Do DSP initialisations */

/* Bind to the Oculus Driver */
   if( odxbind() < 1)
      {
      printf("Cannot Access any ODx Driver\n");
      exit(1);
      }

   setdmajor( 0);                  /* Bind to first device */
   if( breset)   reset();          /* Reset Board          */

   /* Select TMS320C40 */
   opr_set( PROCESSOR, 1);
   errno= opr_inq( ERRNO);
   if( errno==IINVAL)
     {
     printf("Cannot select TMS320C40\n");
     exit( 1);
     }

   /* Set DFB, PFB and FFB to 0 */
   opr_set( DFB, 0);
   opr_set( PFB, 0);
   opr_set( FFB, 0);
   fwin( 0, 0, opr_inq( MW), opr_inq( MH));
   pwin( 0, 0, opr_inq( MW), opr_inq( MH));

/* Zooooooom..... make image bigger on VGA*/
   vidout(0,4,4,0,0);
```

```
 /* wait for it.....for image to appear on VGA */
    printf("Just one second then hit a key.\n");
    getch();

    loop = 0;    /* counter */
    OPTIONS();  /* find out what we want to do */

  /* THE MAIN PROGRAM LOOP STARTS HERE
  counted by loop */

    while(1)
       {
       loop = loop + 1;
       if(kbhit()) OPTIONS();

/* if keyboard is hit go find out what is wanted */

/* now serious business */
/* if bgrab is 1 then grab an image */

       if( bgrab)
          {
          long watchdog;

       /* Grab one frame */
          opr_set( PROCESSOR, 1);
          fbgrab( 1);
          watchdog= 1024;
          while( opr_inq (FGON))
             {
             --watchdog;
             if( watchdog <= 0 )
                {
printf("Unable to grab one frame,
                 so process content of current frame buffer.\n");
                break;
                }
             }
          }

    /* Select the DSP as the current so next ODX functions will be routed to
       the DSP */
       opr_set( PROCESSOR, 1);

  /* option is 0 then do things with image */

       if( option == 0)
          {
          USERFCT0 user0;
```

```
        int x, y;
        int i;

    /*  printf("Draw hexagon coords in PFB\n");  */

      /* transfer arrays to DSP*/
        user0.gArray = (unsigned long far *) Array;
        user0.gArray2 = (unsigned long far *) Array2;
        user0.gCoeffs = (unsigned long far *) Coeffs;
        user0.gCoeff2 = (float far *) Coeff2;
        user0.gCentres = (float far *) Centres;
      /* Write hexagon coords  */
        user0.dir = 0;        /* Stay  */
        user0.grl = 0xff;     /* Gray Level to write  */
        user0.pfbpage = opr_inq( PFBPAGE);
        user0.spotson = diagnostics;
/*Off to the DSP board*/
        if(diagnostics == 1)   printf("Off to DSP!\n");

        USER_DSP_FUNCTION( 0, 0, &user0);

        if(diagnostics == 1)    printf("Back to GSP!\n");
/*check data returned from DSP*/

        if(initialise == 1 && initial2 == 0)
           {
           for(i=0;i<MODES;i++)
              {
              fudge[i] = Coeff2[i];
              initial2 = 1;
              printf("%f\n", fudge[i]);
              }
           }




        for(i=0; i<MODES; i++)
           {
/* fudge the coeffs to remove nonuniformities */
           Coeff2[i] = Coeff2[i] - fudge[i];

           if(diagnostics == 1&& i <10) printf("Frame %d ",loop);
  if(diagnostics == 1 && i< 10) printf("Mode %d is %2.2f\n", i+1, Coeff2[i]);
           if(file_out == 1) fprintf(fpout,"Frame %d ",loop);
  if(file_out == 1) fprintf(fpout,"Mode %d is %2.2f\n", i+1, Coeff2[i]);

           }
```

```
   /* Display the centre data */
         for(i=0; i<POINTS*2-110; i=i+2)
            {
            printf("at pixel %d coord %ld %ld dx and dy are %f %f\n", i/2,
 Array[i], Array[i+1], Centres[i], Centres[i+1]);
            }

 if(diagnostics == 1) printf("buffer2 is %ld %ld\n",Array2[i], Array2[i+1]);


         if(diagnostics == 1)
            {
            for(delay=0;delay<10000;delay++)
               {
               for(delay2=0;delay2<300;delay2++){}
               }
            }

 /* recentre the hex coords with the lenslet spots without any
       aberation, ie xcen ycen alters hexcoords */


         if(initialise == 0)
            {
            for(i=0;i<POINTS*2;i=i+2)
               {

               printf("%ld %ld ",Array[i], Array[i+1]);
               Array[i] = Array[i]+(int)(Centres[i]+0.5);
               Array[i+1] = Array[i+1]+(int)(Centres[i+1]+0.5);
 printf("corrected to %ld %ld for pixel %d, ",Array[i], Array[i+1], i/2);



               }
            }
            initialise = 1;


         }       /* end of option */



 /******correction********/
       if(correct == 1)
          {


    /*    coeffs[1 ]=-GAIN1*Coeff2[2 ];
```

```
        coeffs[2 ]=-GAIN2*Coeff2[ ];  */
   /* No tip-tilt */

 /* n.b. these all depend on device orientation in system */
 /* THIS MUST BE CHECKED AFTER OPTICAL SYSTEM SETUP */

        if(diagnostics == 1) printf("old mode 3 is %f\n", coeffs[3]);
        coeffs[3]= coeffs[3] - (GAIN3 * Coeff2[3-1]);
        if(coeffs[3] > 0.5) coeffs[3] = coeffs[3] + (GAIN3 * Coeff2[3-1]);
        if(coeffs[3] < -0.5) coeffs[3] = coeffs[3] + (GAIN3 * Coeff2[3-1]);
        if(diagnostics == 1)
printf("new corrected mode 3 is %f,
when measured coeff is %f and gain is %f\n",
coeffs[3], Coeff2[3-1], GAIN3);

        coeffs[4 ]= coeffs[4] - GAIN4*Coeff2[4-1 ];
        if(coeffs[4]>0.5) coeffs[4] = coeffs[4] + (GAIN4 * Coeff2[4-1]);
        if(coeffs[4]<-0.5) coeffs[4] = coeffs[4] + (GAIN4 * Coeff2[4-1]);
        if(diagnostics == 1)
printf("new corrected mode 4 is %f,
when measured coeff is %f and gain is %f\n",
coeffs[4], Coeff2[4-1], GAIN4);

        coeffs[5 ]=coeffs[5] + GAIN5*Coeff2[5-1 ];
        if(coeffs[5]>0.5) coeffs[5] = coeffs[5] - (GAIN5 * Coeff2[5-1]);
        if(coeffs[5]<-0.5) coeffs[5] = coeffs[5] - (GAIN5 * Coeff2[5-1]);
        if(diagnostics == 1)
printf("new corrected mode 5 is %f,
when measured coeff is %f and gain is %f\n",
coeffs[5], Coeff2[5-1], GAIN5);

        coeffs[6 ]=coeffs[6] + GAIN6*Coeff2[6-1 ];
        if(coeffs[6]>0.5) coeffs[6] = coeffs[6] - (GAIN6 * Coeff2[6-1]);
        if(coeffs[6]<-0.5) coeffs[6] = coeffs[6] - (GAIN6 * Coeff2[6-1]);

        coeffs[7 ]=coeffs[7] - GAIN7*Coeff2[7-1 ];
        if(coeffs[7]>0.5) coeffs[7] = coeffs[7] + (GAIN7 * Coeff2[7-1]);
        if(coeffs[7]<-0.5) coeffs[7] = coeffs[7] + (GAIN7 * Coeff2[7-1]);


        coeffs[8 ]=coeffs[8] - GAIN8*Coeff2[8-1 ];

        coeffs[9 ]= coeffs[9] + GAIN9*Coeff2[10-1 ];

        coeffs[10 ]= coeffs[10] - GAIN10*Coeff2[9-1 ];

        coeffs[11 ]= coeffs[11] - GAIN11*Coeff2[11-1 ];

        coeffs[12 ]= coeffs[12] + GAIN12*Coeff2[12-1 ];
```

```
/* etc. etc. */

/* send coeffs to SLM routines */
        lc_go(coeffs);
        }   /* end of if correct */
    }      /* end of while kbdhit */

  } /* end of main */

/* subroutines */


void OPTIONS(void)
{
  int answer;
  printf("            OPTIONS\n");
  printf("1. Align System\n");
  printf("2. Measure modes,
       no correction, align and fudge (run before 3,4,5)\n");
  printf("3. Measure modes, no correct, no diagnostics\n");
  printf("4. Measure modes and correct with diagnostics\n");
  printf("5. Measure modes and correct without diagnostics\n");
  printf("6. Measure modes and correct - diagnostics to file\n");
  printf("7. Quit\n");

  scanf("%i", &answer);

  switch(answer)
  {
    case 1:
      printf("Align system\n");
      initialise = 1;  /* don't initialise */
      initial2 = 1;    /* don't fudge */
      diagnostics = 1; /* diagnostics on */
      correct = 0;     /* no correction */
      file_out = 0;   /* no data out to file */
    break;

    case 2:
      printf("Measuring modes\n");
      initialise = 0;  /* initialise */
      initial2 = 0;    /* fudge */
      diagnostics = 1; /* diagnostics on */
      correct = 0;     /* no correction */
      file_out = 0;   /* no data out to file */
```

```
        break;

        case 3:
           printf("Only measuring\n");
           initialise = 1;
           initial2 = 1;
           diagnostics = 0;
           correct = 0;
           file_out = 0;
        break;

        case 4:
           printf("Correcting\n");
           initialise = 1;  /* initialise */
           initial2 = 1;    /* fudge */
           diagnostics = 1; /* diagnostics on */
           correct = 1;     /* correction */
           file_out = 0;   /* no data out to file */
        break;

        case 5:
           printf("Correcting\n");
           initialise = 1;  /* initialise */
           initial2 = 1;    /* fudge */
           diagnostics = 0; /* diagnostics off */
           correct = 1;     /* correction */
           file_out = 0;   /* no data out to file */
        break;

        case 6:
           printf("Correcting and output to file out.dat\n");
           initialise = 1;  /* initialise */
           initial2 = 1;    /* fudge */
           diagnostics = 0; /* diagnostics off */
           correct = 1;     /* correction */
           file_out = 1;   /* data out to file */
        break;

        case 7:
           printf("Bye!\n");
           exit(0);

        default:
           printf("\n No change!\n");

        }
}
```

E Software Listings: test_dsp.c

```
/**

Zernike measurement from shack-hartmann spots
hex array geometric and then recentred

J.Gourlay
1 Nov 1996

stuff from coreco follows......

   History
      1.00 July 10th, 1994, JPC

      1.01 March 5th, 1994, JPC
         Add new demo user functions showing how to:
            manipulate frame buffer using cvfbxyl function
            perform DMA transfers
            use communication ports

$$
Name:   ODXUSER

Declaration:

Input:
         wLib:       library number (valid range [ 0 - 15])
         wOperation: function number to execute (valid range [0 - 2048])
         buf:        far pointer to user defined structure

Output:  NONE

Return Value:
         user defined return value

Description:
      This function is mapped in entry #47 of the ODX Functions.
      It is a general user defined function.

      Each function must have the following template:

         msg_copy( GspToDsp, ...)      (optional: copy data from gsp to dsp)
         msg_wait()
            .
            . any code
            .
         msg_copy( DspToGsp...)        (optional: copy data from dsp to gsp)
            .
            . any code
            .
```

```
        msg_send(x)

                ------------------------
      Example Functions

      Function #0
         This function performs a write pixel.

      Function #1
         This function simply returns the error value ERRNO.

      Function #2
         This function convolves an image with a user-defined kernel.

      Function #3
         This function computes an histogram.
**/
#include "intpt40.h"
#include "dma40.h"
#include "setjmp.h"
#include "user.h"
#include "opr.h"


void odxuser( unsigned short wLib, unsigned short wOperation, GSPADDR buf)
   {
   /* Remember: wOperation must range from 0 to 2048 and wLib from 0 to 15. */
   /* If not, this function will not be called. */
   /* The unused index are reserved for internal use. */
   switch( wLib)
      {
      case 0:
         {
         void UserLib_0( unsigned short wOperation, GSPADDR hBuf);
         UserLib_0( wOperation, buf);
         }
         break;

       /* Undefined Library */
      default:
         msg_wait();
         opr_set( ERRNO, INOENT);
         /* End of processing */
         msg_send( 0);
         break;
      }
   return;
   }
```

```
/* Don't forget: For the TMS320C40 all C data types are 32 bits */
/* That's why sizeof( USERFCT0) would return 5 */

/* Check this corresponds to test_gsp.c structure!!!!! */

typedef struct
   {
   unsigned long spoton;
   unsigned long xpos;
   unsigned long ypos;
   unsigned long dir;
   unsigned long grl;
   unsigned long pfbpage;
   GSPADDR gArray;
   GSPADDR gArray2;
   GSPADDR gCoeffs;
   GSPADDR gCoeff2;
   GSPADDR gCentres;

   }  USERFCT0;

typedef struct
   {
   GSPADDR gMask;
   unsigned long wXs;
   unsigned long wYs;
   } USERFCT2;

typedef struct
   {
   /* Address of buffer in GSP (HOST) memory */
   GSPADDR gBuffer;
   } USERFCT3;

typedef struct
   {
   GSPADDR gMask;
   unsigned long wXs;
   unsigned long wYs;
   } USERFCT8;

void UserLib_0( unsigned short wOperation, GSPADDR buf)
   {
   switch( wOperation)
      {

/* we will use case 0 for our Shack Hartmann measurements */
      case 0:
         {
```

```
        DSPADDR pix8;
        USERFCT0 userfct0;
        unsigned int uCurrentPfbpage;
        int *buffer,*buffer2,*Coeffs;
        float Coeff2[12];
        float Centres[38];
        int i,x,y;
        unsigned rdpixel;
        unsigned int *pixel,*start;
        int box=7; /* centroid sample box size */
        float x_cen,y_cen,total_flux;
        int flash;
        int delay;

/* Interaction definition - calculated offline with matlab 19 lenslets, 12 modes*/
        float i_matrix[12][38] =  {
{ 0.0935047, 5.5103e-009, 0.0692363, 6.74936e-009, 0.0813705, -0.00700567, 0.0813705,
0.00700568, 0.0692363, 1.51496e-008, 0.0813704, -0.00700569, 0.0813705, 0.00700567,
-0.00356889, 8.3699e-009, 0.0328336, -0.0210171, 0.0449679, -0.0280227, 0.0692363,
-1.54955e-008, 0.0449679, 0.0280227, 0.0328336, 0.0210171, -0.00356889, 4.61641e-008,
0.0328336, -0.021017, 0.0449678, -0.0280227, 0.0692363, -5.0798e-008, 0.044968, 0.0280227,
0.0328337, 0.0210171},
{ 5.5103e-009, 0.0935047, 1.16494e-008, 0.0854152, -0.00700566, 0.073281, 0.00700569,
0.073281, 1.45418e-008, 0.0854152, -0.0070057, 0.073281, 0.00700566, 0.073281,
1.49669e-008, 0.0611469, -0.021017, 0.057102, -0.0280227, 0.01261, -1.33283e-009,
0.0206994, 0.0280227, 0.01261, 0.0210171, 0.057102, 5.67357e-008, 0.0611468, -0.021017,
 0.0571021, -0.0280228, 0.0126101, -7.78373e-008, 0.0206994, 0.0280227, 0.01261, 0.0210171,
 0.057102 },
{ 3.66981e-009, -2.39091e-009, 0.0416667, -6.39599e-009, 0.0208334, 0.0360844, -0.0208334,
 0.0360845, -0.0416667, 2.81788e-008, -0.0208334, -0.0360844, 0.0208333, -0.0360845,
1.04074e-008, -7.14246e-009, 0.0208333, 0.0120281, -7.59731e-009, -1.63658e-008,
4.99681e-009, 0.0240562, 1.10395e-008, -1.5162e-008, -0.0208333, 0.0120281,
6.41105e-009, -3.12864e-010, -0.0208333, -0.0120281, 1.01043e-008, 1.18262e-008,
 -1.82671e-008, -0.0240562, -3.90188e-009, 2.19085e-008, 0.0208333, -0.0120281},
{ 5.90697e-009, 1.04791e-008, 0.0433647, 1.79923e-008, 0.0275601, -0.0409484, -0.0275601,
 -0.0409484, -0.0433647, -3.31415e-008, -0.0275601, 0.0409484, 0.02756, 0.0409485,

-0.00731451, 2.38894e-008, 0.0356583, -0.0409484, 0.0433647, -0.0208136, 3.46502e-008,
-0.0208135, -0.0433647, -0.0208135, -0.0356583, -0.0409485, 0.00731455, -7.01868e-008,
 -0.0356582, 0.0409484, -0.0433647, 0.0208136, -1.16022e-007, 0.0208135, 0.0433647,
 0.0208135, 0.0356583, 0.0409485 } ,
{ 1.03101e-009, 1.71118e-008, 2.14683e-009, 0.0512017, 0.0409485, 0.0197231, 0.0409485,
-0.0197231, 2.99395e-008, -0.0512017, -0.0409484, -0.0197231, -0.0409485, 0.0197231,
2.07929e-008, 0.0553815, 0.000226167, 0.0356583, 0.0208135, -0.0193313, 0.0615358,

-1.10214e-008, 0.0208136, 0.0193313, 0.000226192, -0.0356582, -3.72743e-008, -0.0553814,
-0.00022614, -0.0356583, -0.0208135, 0.0193312, -0.0615358, 3.03528e-008, -0.0208136,

-0.0193313, -0.000226195, 0.0356582 },
```

```
{ -0.0228408, 2.13774e-010, -0.00927909, 2.88831e-010, -0.01606, 0.00391493, -0.01606,
 -0.00391493, -0.00927908, -9.70836e-009, -0.0160599, 0.00391494, -0.01606, -0.00391493,
 0.0314061, -9.58313e-009, 0.0110635, 0.0117448, 0.00428264, 0.0156597, -0.00927908,
9.47197e-009, 0.00428263, -0.0156597, 0.0110635, -0.0117448, 0.0314061, -2.93776e-008,
0.0110636, 0.0117448, 0.00428267, 0.0156598, -0.00927908, 2.98226e-008, 0.00428262,
-0.0156597, 0.0110635, -0.0117448 } ,
{ 2.13774e-010, -0.0228408, -1.28237e-009, -0.0183203, 0.00391493, -0.0115394, -0.00391493,
-0.0115394, -8.2137e-009, -0.0183202, 0.00391494, -0.0115394, -0.00391492, -0.0115394,
-1.48265e-008, -0.00475853, 0.0117448, -0.0024982, 0.0156597, 0.0223649, 7.27433e-009,
0.0178444, -0.0156597, 0.022365, -0.0117448, -0.00249818, -2.44404e-008, -0.00475853,
0.0117448, -0.00249821, 0.0156597, 0.0223649, 3.22498e-008, 0.0178444, -0.0156597,
0.022365, -0.0117448, -0.00249817 }  ,
{ -2.42858e-009, 1.43843e-009, -0.0166667, 3.17586e-009, -0.00833335, -0.0144338,
0.00833334, -0.0144338, 0.0166667, -1.23842e-008, 0.00833336, 0.0144338, -0.00833334,
0.0144338, 0.0111111, -1.26029e-010, -0.00277775, -0.00160374, 0.00555555, 0.00962249,
 -1.79148e-010, -0.00320747, -0.00555555, 0.0096225, 0.00277775, -0.00160374,
-0.0111111, 5.49695e-009, 0.00277775, 0.00160374, -0.00555556, -0.00962249,
3.72831e-010, 0.00320747, 0.00555555, -0.00962251, -0.00277775, 0.00160374 } ,
{ -1.57475e-008, 7.77532e-009, 0.008547, 1.48212e-008, -0.00427349, -0.00740192,
 -0.00427351, 0.00740191, 0.00854699, 9.98798e-009, -0.00427351, -0.00740192,
-0.00427352, 0.00740192, 0.034188, 9.18346e-009, 0.0128205, -0.0222058, -0.017094,
-0.0296077, -0.025641, -1.67923e-008, -0.017094, 0.0296077, 0.0128205, 0.0222058,
0.034188, 4.34013e-008, 0.0128205, -0.0222058, -0.0170939, -0.0296077, -0.025641,
-5.19173e-008, -0.017094, 0.0296076, 0.0128205, 0.0222058 } ,
{ 5.23938e-009, 1.98603e-008, 5.25102e-009, 0.00854701, 0.00740193, -0.00427349,
-0.00740192, -0.00427349, -1.56579e-008, 0.00854702, 0.00740193, -0.00427347,
-0.0074019, -0.00427349, -3.75212e-008, 0.034188, 0.0222058, 0.0128205, 0.0296077,
 -0.017094, 1.70102e-008, -0.025641, -0.0296077, -0.017094, -0.0222058, 0.0128205,
-3.55425e-008, 0.034188, 0.0222058, 0.0128205, 0.0296077, -0.017094, 5.07903e-008,
-0.0256411, -0.0296077, -0.017094, -0.0222058, 0.0128205 }      ,
{ -2.80025e-009, -1.43685e-009, -0.0120167, -2.45854e-009, -0.00914316, 0.0122167,
 0.00914315, 0.0122167, 0.0120167, 9.55247e-009, 0.00914316, -0.0122167, -0.00914314,
-0.0122167, 0.0261233, -1.08381e-008, -0.00235108, 0.0122167, -0.0120167, -0.0081444,
-1.03601e-008, -0.00814449, 0.0120167, -0.00814444, 0.00235109, 0.0122167, -0.0261233,
2.11996e-008, 0.00235105, -0.0122167, 0.0120167, 0.00814438, 3.81383e-008, 0.0081445,
-0.0120167, 0.00814447, -0.0023511, -0.0122167 }  ,
{ 2.01596e-009, -4.89757e-009, 1.63434e-009, -0.0161965, -0.0122167, -0.00496344,
-0.0122167, 0.00496343, -9.92387e-009, 0.0161965, 0.0122167, 0.00496344, 0.0122167,
-0.00496343, -2.55041e-008, -0.00731455, 0.00950187, -0.00235107, 0.00814444, 0.0214211,
-0.0135741, 1.06266e-008, 0.0081444, -0.0214211, 0.00950186, 0.00235105, 3.96675e-008,
0.00731455, -0.00950187, 0.0023511, -0.00814446, -0.0214211, 0.0135741, -3.55869e-008,
-0.00814439, 0.0214211, -0.00950188, -0.00235103 }
};

        /* Read  Parameters ( parameter size must be specified in bytes */
        /* so you must multiply by 4 the value return by the sizeof function) */

/* data from gsp */
     msg_copy( GspToDsp, buf, (DSPADDR)&userfct0, 4 * sizeof( USERFCT0));
```

```
/* important - convertion for floats */
      IEEEtoC40( Coeff2, Coeff2, 12);

/*transfer data to buffer*/
      buffer = (int *) s_malloc(19*2);
      if(buffer==0){
        opr_set(ERRNO, INOSPC);
        msg_send(0);
        return;
        }
      buffer2 = (int *) s_malloc(19*2);
      if(buffer2==0){
        opr_set(ERRNO, INOSPC);
        msg_send(0);
        return;
        }
      Coeffs = (int *) s_malloc(12);
      if(Coeffs==0){
        opr_set(ERRNO, INOSPC);
        msg_send(0);
        return;
        }

/*get lenslet centre points from GSP(Host)*/
      msg_copy(GspToDsp, userfct0.gArray, (DSPADDR) buffer, 4*2*19);

         /* Gsp waits */
      msg_wait();

         /* End of processing */
         /*msg_send(0);*/

         /* Do Operations while Gsp is released   */
         /* Save Current Page and select user page */
         uCurrentPfbpage = opr_inq( PFBPAGE);
         opr_set( PFBPAGE, userfct0.pfbpage);

         /* Move to pixel position  */
         /*amov( userfct0.xpos, userfct0.ypos); */

         /* Access frame buffer */

         /*get start address of frame*/

      pix8=cvfbxyl(0,0,0);
```

```
      /*allocate memeory for image*/
       start = (unsigned int *) s_malloc(16384);
       if( start==0)
          {
            opr_set( ERRNO, INOSPC);
            msg_send( 0);
            return;
          }

/* Allocated temporary buffer in static ram */

          /*loop through the frame-slow but simple*/
         for(y=0;y<128; y++){
                for(x=0;x<128/4;x++){

                        rdpixel=*pix8;
                        (*start++)= (rdpixel & 0xff);
                        (*start++)= (rdpixel & 0xff00)>>8;
                        (*start++)= (rdpixel & 0xff0000)>>16;
                        (*start++)= (rdpixel & 0xff000000)>>24;
                        *pix8++;
                }
                pix8=pix8+opr_inq(MW)/4-128/4;
          }
          start = start - 16384;

         /* Find centroid of point */

         for(i=0;i<1*38;i=i+2){
                total_flux=0.0;
                x_cen=0.0;
                y_cen=0.0;
                for(y=1;y<box*2;y++){
                        for(x=1;x<box*2;x++){
 x_cen=x_cen+ (float) x*(*(start+x+buffer[i]-box+y*128+buffer[i+1]*128-box*128));
 y_cen=y_cen+ (float) y*(*(start+x+buffer[i]-box+y*128+buffer[i+1]*128-box*128));
total_flux=total_flux+ (float) (*(start+x+buffer[i]-box+y*128+buffer[i+1]*128-box*128));
                        }
                }

                /* Normalise the centroid locations */
                /* and write to buffer*/
                Centres[i] = x_cen / total_flux - (float)box;
                Centres[i+1] = y_cen / total_flux - (float)box;


                /* put point at zeroth location */
                if(userfct0.spoton == 1)
                  {
```

```
                amov(buffer[i], buffer[i+1]);
                wtpix(0xff,0);
                for(delay=0;delay<10000;delay++){};
/* flash a dark spot at the centroid locations */
                amov(buffer[i]+(int)(Centres[i]+0.5), buffer[i+1]+(int)(Centres[i+1]+0.5));
                wtpix(0x00,0);
                for(delay=0;delay<10000;delay++){};

                }
          }   /* end of loop through spots */


        /* matrix multiply */


        for(x=0;x<12;x++){
              Coeff2[x] = 0;
              for(y=0;y<38;y++){
                    Coeff2[x]=Coeff2[x]+Centres[y]*i_matrix[x][y];
                    }
              }

       /* test floating point stuff */

        /* convert float back */
        C40toIEEE( Coeff2, Coeff2, 12);
        C40toIEEE ( Centres, Centres, 2*19);
        /* Restore Page   */
        opr_set( PFBPAGE, uCurrentPfbpage);

        /* send coefficients data back to GSP*/
          msg_copy( DspToGsp, userfct0.gCoeff2, (DSPADDR) Coeff2, 4*12);
          msg_copy( DspToGsp, userfct0.gCoeffs, (DSPADDR) Coeffs, 4*12);
          msg_copy( DspToGsp, userfct0.gArray2, (DSPADDR) buffer2, 4*2*19);

          msg_copy( DspToGsp, userfct0.gCentres, (DSPADDR) Centres, 4*2*19);
         /* End of processing */
        msg_send(0);

        /*free memory*/
        s_free(buffer);
        s_free(buffer2);
        s_free(Coeffs);
        s_free(start);
        }
        return;

/* you can just forget about all the following crap */
```

```
/* This function simply returns the error value ERRNO */
case 1:
   /* Gsp waits */
   msg_wait();

   /* End of processing */
   msg_send( opr_inq( ERRNO));
   return;

case 2:
   {
   USERFCT2 userfct2;
   int i, j;
   float anMask[ 15*15+1];
   int *kernel;
   int value, nelement, nval;

   /* Read  Parameters ( parameter size must be specified in bytes */
   /* so you must multiply by 4 the value returned by the sizeof function) */

   /* Gets information from GSP (HOST) */
   msg_copy( GspToDsp, buf, (DSPADDR)&userfct2, 4 * sizeof( USERFCT2));

   if( userfct2.wXs > 15 || userfct2.wYs > 15)
      {
      opr_set( ERRNO, IINVAL);
      msg_send( 0);
      return;
      }
   nelement= userfct2.wXs*userfct2.wYs;

   /* Allocate space where will be stored kernel values */
   kernel= (int *) s_malloc( nelement + 1);
   if( kernel == 0)
      {
      opr_set( ERRNO, INOSPC);
      msg_send( 0);
      return;
      }

   /* Gets kernel values from GSP (HOST) */
   msg_copy( GspToDsp, userfct2.gMask, (DSPADDR) kernel, 2 * (nelement + 1));

   /* GSP waits */
   msg_wait();

   /* Convert kernel to floating point values */
   i= 0; j= 0;
   while( 1)
```

```
      {
      value= kernel[ j++];
      nval= value & 0xffff;
      if( nval & 0x8000) nval|= 0xffff0000;
      anMask[ i++]= (float) nval;
      value >>= 16;
      nval= value & 0xffff;
      if( nval & 0x8000) nval|= 0xffff0000;
      if( i >= nelement) break;
      anMask[ i++]= (float) nval;
      }
   if ( nval!=0)      anMask[ i]= 1.0/nval;
   else               anMask[ i]= 1.0;

   /* Free buffer */
   s_free( kernel);

   /* Convolve processing window of PFB with the mask */
   convo( anMask, userfct2.wXs, userfct2.wYs);
   }
   break;

case 3:
   {
   USERFCT3 userfct3;
   unsigned int *auHisto;
   unsigned int uN;

   /* Gets information from GSP (HOST) */
   msg_copy( GspToDsp, buf, (DSPADDR)&userfct3, 4 * sizeof( USERFCT3));
   /* Gsp waits */
   msg_wait();

   /* Allocated temporary buffer in static ram */
   auHisto= (unsigned int *) s_malloc( 4096);
   if( auHisto==0)
      {
      opr_set( ERRNO, INOSPC);
      msg_send( 0);
      return;
      }
   /* Gets histogram for processing window of pfb */
   uN= histo( auHisto, 0, 8, 0);
   /* Send result to GSP (HOST) */
   msg_copy( DspToGsp, userfct3.gBuffer, (DSPADDR)auHisto, 4 * uN);
   /* Free buffer */
   s_free( auHisto);
   }
   break;
```

```
case 4:
/* Exemple that demonstrates how to access the video frame buffer */
/* from the DSP processor                                          */
/* For this exemple, we will only replace each pixel grey level by
   its negation:  0 will become 255
                  1 will become 254
                  ...
                  255 will become 0
 */
/* GSP waits */
msg_wait();

if( opr_inq( PIXSIZ) == 8)
   {
   /* We are in 8 bits/pixel mode */
   DSPADDR pix8;
   unsigned rdpixel, wtpixel;
   unsigned x, y;

   /* First, uses cvfbxyl function to get the address of the
      current processing frame buffer and window.
   */
   pix8= cvfbxyl(0,0,0); /*address of first pixel*/
   /* Next, process the processing window */
   /* For each line of pwin */
   for( y= 0 ; y < opr_inq( PWIN_YLEN); y++)
      {
      /* Since the DSP performs 32 bits access,
         each reading gets 4 pixels from video ram when in 8 bits/pixel
      */
      /* For each group of 4 pixels of the current line */
      for( x= 0; x < (opr_inq( PWIN_XLEN)/4); x++)
         {
         /* Read 4 pixel from video ram */
         rdpixel= *pix8;

         /* Negate each pixel */
         /* Could have been written more simply like this */
         /* wtpixel= ~rdpixel; */
         /* But for the purpose of this exemple   */
         /* we will process each pixel separately */
         rdpixel= ~rdpixel;
         wtpixel=  (rdpixel & 0xff);
         rdpixel>>= 8;
         wtpixel|= (rdpixel & 0xff) << 8;
         rdpixel>>= 8;
         wtpixel|= (rdpixel & 0xff) << 16;
         rdpixel>>= 8;
```

```
            wtpixel|= (rdpixel & 0xff) << 24;

            /* Write back result in video ram */
            *pix8++= wtpixel;
            }
         /* Update video frame buffer address */
         pix8+= (opr_inq( PMW) - opr_inq( PWIN_XLEN)) >> 2;
         }
      }
   else
      {
      /* We are in 16 bits/pixel mode */
      DSPADDR pix16;
      unsigned rdpixel, wtpixel;
      unsigned x, y;

      /* First, uses cvfbxyl function to get the address of the
         current processing frame buffer.
      */
      pix16= cvfbxyl( opr_inq( PFB), opr_inq( PWIN_XMIN), opr_inq( PWIN_YMIN));
      /* Next, process the processing window as defined by pwin structure */

      /* For each line of pwin */
      for( y= 0 ; y < opr_inq( PWIN_YLEN); y++)
         {
         /* Since the DSP performs 32 bits access,
            each reading gets 2 pixels from video ram when in 16 bits/pixel
         */
         /* For each group of 2 pixels of the current line */
         for( x= 0; x < (opr_inq( PWIN_XLEN)/2); x++)
            {
            /* Read 2 pixel from video ram */
            rdpixel= *pix16;

            /* Negate each pixel */
            /* Could have been written more simply like this */
            /* wtpixel= ~rdpixel; */
            /* But for the purpose of this exemple   */
            /* we will process each pixel separately */
            rdpixel= ~rdpixel;
            wtpixel=  (rdpixel & 0xffff);
            rdpixel>>= 16;
            wtpixel|= (rdpixel & 0xffff) << 16;

            /* Write back result in video ram */
            *pix16++= wtpixel;
            }
         /* Update video frame buffer address */
         pix16+= (opr_inq( PMW) - opr_inq( PWIN_XLEN)) >> 1;
```

```
      }
    }
    break;

case 5:
    {
    /* Do not call this function if communication port 1 and 3 of your F64 board
       are not properly connected. */
    /* Show how to use DMA controller, communication ports and interrupt
       vector table (IVTP) */

    extern void DMASendToCommPort( int ch);          /* See function below */
    extern void DMAReceiveFromCommPort( int ch);     /* See function below */

    /* Gsp waits */
    msg_wait();

    /* Send processing frame buffer (PFB)  to comm. port 1 */
    DMASendToCommPort( 1);
    /* Receive data transfered to comm. port 1 using comm. port 3
       Write received data in DFB */
    DMAReceiveFromCommPort( 3);
    }
    break;

/* Show how to copy processing window from FFB to PFB */
case 6:
    {
    unsigned i;  /* loop index */
    /* Initialize xlen, ylen to processing window size */
    unsigned xlen=   opr_inq( PWIN_XLEN);
    unsigned ylen=   opr_inq( PWIN_YLEN);
    /* Initialize offset to offset in pixel between two lines */
    unsigned offset= opr_inq( MW)-opr_inq( PWIN_XLEN);
    /* Get starting address of PFB ( could be any address in DSP memory space) */
    DSPADDR pdst= cvfbxyl( opr_inq( PFB), opr_inq( PWIN_XMIN), opr_inq( PWIN_YMIN));
    /* Get starting address of acquisition window of FFB ( could be .. in DSP memory space) */
    DSPADDR psrc= cvfbxyl( opr_inq( FFB), opr_inq( FWIN_XMIN), opr_inq( FWIN_YMIN));

    /* Gsp waits */
    msg_wait();

    if( opr_inq( PIXSIZ) == 8)
       {
       /* In 8 bits mode, each access gets 4 pixels so divide by 4 */
       xlen= xlen/4;
       offset= offset/4;
       }
    else  /* PIXSIZ == 16 */
```

```
        {
        /* In 16 bits mode, each access gets 2 pixels so divide by 2 */
        xlen= xlen/4;
        offset= offset/4;
        }

     /* Transfer each line */
     while( ylen--)
        {
        /* Copy one line */
        for( i= 0; i < xlen; i++) *pdst++= *psrc++;
        /* Add offset to each pointer */
        psrc+= offset;
        pdst+= offset;
        }
     }
     break;

/* Show how to move data using DMA coprocessor */
case 7:
   {
   extern void MoveFrameBufferUsingDMA( void);

   /* GSP waits */
   msg_wait();
   MoveFrameBufferUsingDMA();
   }
   break;

/* Convolution using a floating point kernel sent by the host */
case 8:
   {
   USERFCT8 userfct8;
   float anMask[ 15*15+1];
   int nelement;
   int i;

   /* Read  Parameters ( parameter size must be specified in bytes */
   /* so you must multiply by 4 the value returned by the sizeof function) */

   /* Gets information from GSP (HOST) */
   msg_copy( GspToDsp, buf, (DSPADDR)&userfct8, 4 * sizeof( USERFCT8));

   if( userfct8.wXs > 15 || userfct8.wYs > 15)
      {
      opr_set( ERRNO, IINVAL);
      msg_send( 0);
      return;
      }
```

```
            nelement= userfct8.wXs*userfct8.wYs;

            /* Gets kernel values from GSP (HOST) */
            msg_copy( GspToDsp, userfct8.gMask, (DSPADDR) anMask, 4 * (nelement + 1));

            /* GSP waits */
            msg_wait();

            /* Convert IEEE floating point kernel to DSP format */
            IEEEtoC40( anMask, anMask, nelement+1);

            /* Convolve processing window of PFB using the kernel */
            convo( anMask, userfct8.wXs, userfct8.wYs);

            /* Return kernel back to host */
            C40toIEEE( anMask, anMask, nelement+1);
            msg_copy( DspToGsp, userfct8.gMask, (DSPADDR)anMask, 4 * (nelement+1));
            }
            break;

         /* Undefined Operation */
         default:
            msg_wait();
            opr_set( ERRNO, INOENT);
            break;
         }
      /* End of processing */
      msg_send( 0);
      }

/*************************************************************************/
/* Uses DMA 1 for moving datas from processing frame buffer (PFB) to display
   frame buffer (DFB)
*/
void MoveFrameBufferUsingDMA( void)
   {
   extern void c_int38( void);  /* See function below */
   /* Gets source address */
   DSPADDR psrc= cvfbxyl( opr_inq( PFB), 0, 0);
   /* Gets destination address */
   DSPADDR pdst= cvfbxyl( opr_inq( DFB), 0, 0);
   /* Gets number of pixel to transfer */
   unsigned count= opr_inq( MW) * opr_inq( MH);

   /* Get transfer size in long word */
   if( opr_inq( PIXSIZ) == 8) count/= 4;
   else                       count/= 2;

   /* Install interrupt vector */
```

```
    install_int_vector( (void*)c_int38, 0x26);

    /* Enable interrupt */
    set_iie( DMA1);

    /* Move data using DMA 1 */
    dma_move( 1, psrc, pdst, count);
    }

/* Interrupt function from DMA 1 (see MoveFrameBufferUsingDMA function) */
void c_int38( void)
   {
   /* Disable interrupt */
   reset_iie( DMA1);
   }

/*************************************************************************/
/*
 * Example of how to use DMA to transfer data through communication ports
 * of F64 board.
 */
/* Uses DMA ch to transfer data to communication port ch */
/* In this particular case, we transfer all processing frame buffer (PFB) */
void DMASendToCommPort( int ch)
   {
   DSPADDR psrc;
   unsigned count;

   /* Gets source address */
   psrc= cvfbxyl( opr_inq( PFB), 0, 0);

   /* Gets number of pixel to transfer */
   count= opr_inq( MW) * opr_inq( MH);

   /* Get transfer size in long word */
   if( opr_inq( PIXSIZ) == 8) count/= 4;
   else                       count/= 2;

    /* Send message through com. port 1 */
   send_msg( ch, psrc, count, 1);
   }

/* Uses DMA ch to receive data from communication port ch      */
/* All data received are written in display frame buffer (DFB) */
void DMAReceiveFromCommPort( int ch)
   {
   DSPADDR pdst;

   /* Gets destination address */
```

```
   pdst= cvfbxyl( opr_inq( DFB), 0, 0);

   /* Receive message from com. port 1 */
   receive_msg( ch, pdst, 1);
   }

/*************************************************************************/
/* Main Function running in the background   */
/* This Function must always be present      */
void DoMain()
   {
   while( 1)
      {
      }
   }
```

Software Listings: Other files

cr.bat compliation file:

```
cl -c -AS /Ot %1.c
link /st:6000 %1+zern_127+odxcall, %1;
```

cc.bat compilation file:

```
cl30 -v40 -x2 -o2 -c -qq %1
```